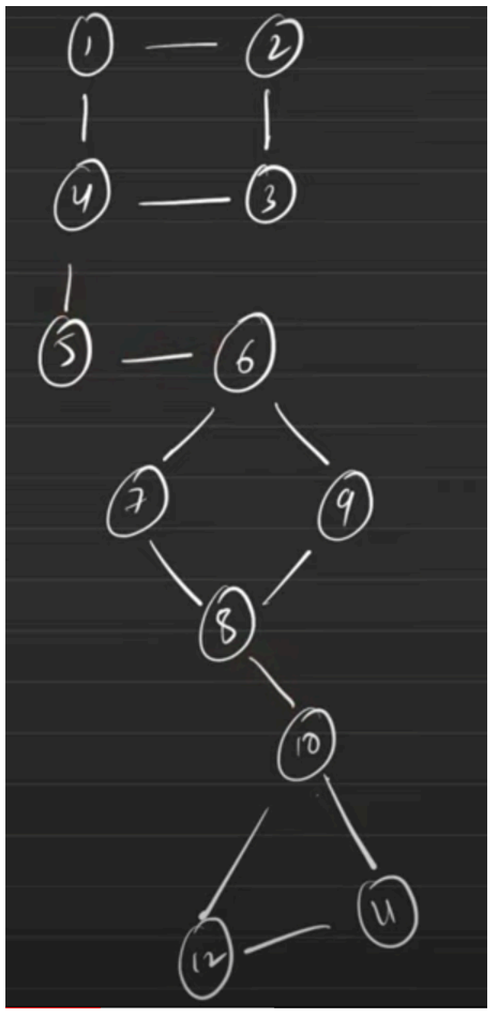
Bridges in graphs are edges which if removed will break the graph into two disconnected components.



One example would be (4)—(5) OR (5)—(6)

We use two arrays ts[] & low[]. ts is time of insertion and low is lowest time of insertion.

ts[] -> DFS time inserts on
low[] -> Min lowest insertion of all adjacent nodes apart from parent.

Time Complexity: O(V+2E), where V = no. of vertices, E = no. of edges. It is because the algorithm is just a simple DFS traversal.

Space Complexity: O(V+2E) + O(3V), where V = no. of vertices, E = no. of edges. O(V+2E) to store the graph in an adjacency list and O(3V) for the three arrays i.e. tin, low, and vis, each of size V.

```java
class Solution {
    // int timer = 1;

    public List<List<Integer>> criticalConnections(int n, List<List<Integer>> connections) {
        List<Integer>[] graph = buildGraph(n, connections);
        List<List<Integer>> bridges = new ArrayList<>();
        boolean[] visited = new boolean[n];
        int[] ts = new int[n];
        int[] low = new int[n];

        dfs(graph, visited, ts, low, bridges, 0, -1, 1);

        return bridges;
    }

    private void dfs(
        List<Integer>[] graph, boolean[] visited, int[] ts, int[] low, List<List<Integer>> bridges,
        int node, int parent, int timer) {

        visited[node] = true;
        ts[node] = low[node] = timer++;

        for (int ng: graph[node]) {
            if (ng == parent) continue;

            if (visited[ng]) {
                low[node] = Math.min(low[ng], low[node]);
                continue;
            }

            dfs(graph, visited, ts, low, bridges, ng, node, timer);
            low[node] = Math.min(low[ng], low[node]);

            if (low[ng] > ts[node]) {
                bridges.add(List.of(ng, node));
            }
        }
    }

    private List<Integer>[] buildGraph(int n, List<List<Integer>> connections) {
        List<Integer>[] graph = new List[n];
        for (int i = 0; i < n; i++)
            graph[i] = new ArrayList<>();

        for (List<Integer> connection: connections) {
            int u = connection.get(0), v = connection.get(1);
            graph[u].add(v);
            graph[v].add(u);
        }

        return graph;
    }
}
```